

Memo on Multi-Slit Emittance Measurement in the FAST/IOTA electron injector

Dean (Chip) Edstrom

Created September, 2015 / Updated February 2018

A multi-slit method for measuring beam emittance was adopted from a Physical Review Special Topics paper¹ and has been implemented in Python. This started as a combination of ACL and Python, (the ACL responsible for peak picking and the python only used for fitting) but all functionality including histogram acquisition has been incorporated into the current version. An initial simulated case, shown below in *fig 1*, was derived from a series of Elegant simulations performed for beam passing through a multi-slit mask at x107 and seen on a screen at x110 ~0.5m downstream. The image was cropped to an 8 x 8 mm square area and inverted. The histogram profile was found in a simple Java program (Appendix A). Since the measurement is amplitude independent (only relative amplitudes are taken into account in the fitting covered later) the histogram was normalized to the highest peak.

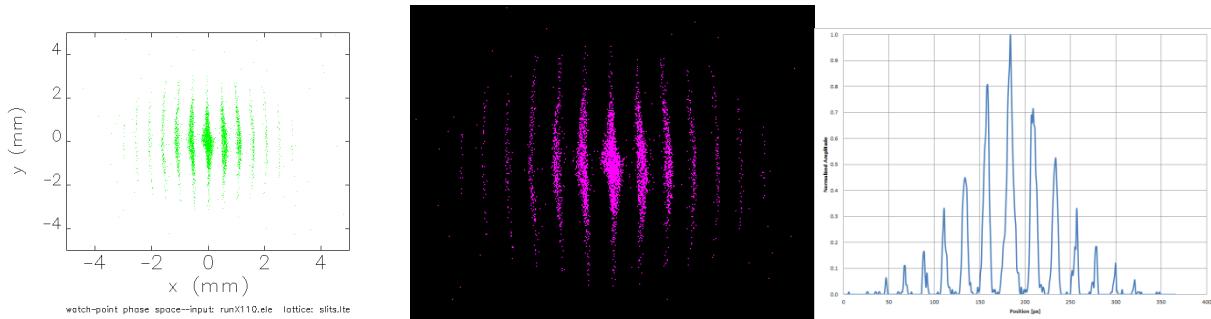


Fig 1 – The original simulated beam image (left, 3200_min_dE.X110.png), the cropped and inverted 8 x 8 mm image (middle), and the normalized horizontal histogram resulting from it (right)

In practice, the histograms for the plane of interest is recorded directly from the camera server histogram array devices, N:DCnXH[0:x] and N:DCnYH[0:y] where n is the camera server being used and 0:x & 0:y are the respective pixel ranges in that dimension. Vertical slits require the horizontal (x) profiles, which allow measurement of the horizontal transverse emittance while horizontal slits allow measurement of the vertical transvers emittances.

The python code `fast_ms_emit.py` (Appendix B), reads the appropriate histogram according to a command line argument. To facilitate integration into other scripts, routines, and sequences various other run-time parameters and options can be specified through the command line as well. If a given parameter is not specified it just uses a default value specified near the top of the python code. Argument support for the multislit emittance measurement is as follows:

General syntax: `python fast_ms_emit.py arg1=val1,arg2=val2,...`

Valid arguments:

```
cam=(1-4)
plane=(X/Y)
aver=(int)
bgsubtract=(0,1,2)
thresh=(float)
```

¹ Space-charge effects in high brightness electron beam emittance measurements, Physical Review Special Topics – Accelerators and Beams, Volume 5, [014201](#) (2002) @ <http://journals.aps.org/prstab/pdf/10.1103/PhysRevSTAB.5.014201>

```

minwidth=(int)
z0=(107/115)
z1=(108/109/111/115/120/124)
res=(float)
lock=(0/1)
plot=(0/1)
bgsubtract=(0/1)
min_peaks=(int)
<FILENAME>

```

The initial distance z_0 in this list is understood to be the location of the multi-slit mask (either at 107 or 115) and z_1 is the location of the YAG screen being monitored. The corresponding distances are derived from survey data but do not take the chicane into consideration. <FILENAME> represents a file name of either a ‘tar-ball’ (.tar) made in performing a measurement or a flat data file with either the formatting of the data file saved in performing a live measurement or a single column list. Note that if it is a single column list, the resolution (res) option should be specified as there is no explicit px/um information included. Distances z_0 and z_1 should also be specified explicitly in the command line.

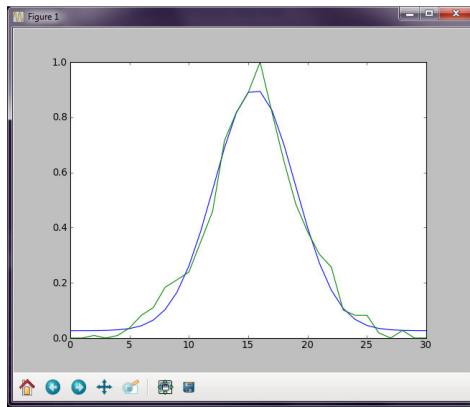


Fig 2 – The Gaussian fit to one of the individual slit beamlets. The Green trace is raw data, the blue is the fit. The x-axis is in units of px.

On initialization, the program will collect the histograms, averaging them together according to the ‘aver’ option. The python code then attempts to pick the number of peaks and perform fits to each of these as seen in Fig. 2. Depending on the size and placement of the beam on the slit mask and transport considerations we can expect this number to be between a single peak and the total number of slits in the mask. A series of steps follow to ultimately obtain the geometric emittance:

- Subtract off the individual fit gaussians and fit what remains to a single gaussian get a background that is then subtracted from the initial data.
- Any offsets for the original individual fits should be zero after this, but the other fit parameters are used to in a single fit to the entire ensemble of gaussians (and the linear offsets for each gaussian are not kept fixed in case there is still some small offset).
- The Integrals are found for each beamlet, m , such that: $I_m = \int_{-\infty}^{\infty} A_m e^{-\frac{1}{2}(\frac{x-x_m}{\sigma})^2} + B_m dx$.
- An envelope fit is then performed. During the 300 MeV commissioning run, the envelope fit was performed using the max of the gaussian fits of the individual gaussian fits to the beamlets, but it was pointed out that

this should probably be updated to use the integral of each gaussian located at the centroid for that beamlet instead.

- Finally, all offsets of the data and fits is made relative to the envelope gaussian fit.

Once these steps are complete, there is enough information available for each beamlet, m , to determine the emittance with the following:

- Beam divergences: $x'_{m,c} = \langle x_m - mw \rangle / L$
- RMS divergence spread: $\sigma'_m = \sqrt{\langle x_m^2 \rangle / L^2 - (x'_{m,c})^2}$
- And the second moments: $\langle x^2 \rangle = \sum_{m=1}^N I_m x_{m,c}^2 / \sum_{m=1}^N I_m$

$$\langle x'^2 \rangle = \sum_{m=1}^N I_m (x'_{m,c}^2 + \sigma'^2_m) / \sum_{m=1}^N I_m$$

$$\langle xx' \rangle = \sum_{m=1}^N I_m x_{m,c} x'_{m,c} / \sum_{m=1}^N I_m$$

Where $\langle x_m \rangle$ is the centroid position for each fit beamlet. The highest peak is $m = 1$, and all other beamlets are arranged around it, increasing outward from there in integer steps. From these, the geometric emittance is:

$$\varepsilon \equiv \sqrt{\langle x^2 \rangle \langle x'^2 \rangle - \langle xx' \rangle^2} = 3.215 \cdot 10^{-2} \mu\text{m}$$

The Twiss parameters are also calculated and the phase ellipse is calculated from these as follows:

$$\alpha = -\frac{\langle xx' \rangle}{\varepsilon}, \quad \beta = \frac{\langle x^2 \rangle}{\varepsilon}, \quad \gamma = \frac{1 + \alpha^2}{\beta}$$

All information is output to a webpage (Appendix D) for display and archived in a tar file for future reference or to re-fit the same data with different parameters (e.g. adjusting the threshold), the file including the composite profile with all averaging performed on collection, the resulting images, and the webpage output including all gaussian information, Twiss parameters, and relevant measurement parameters.

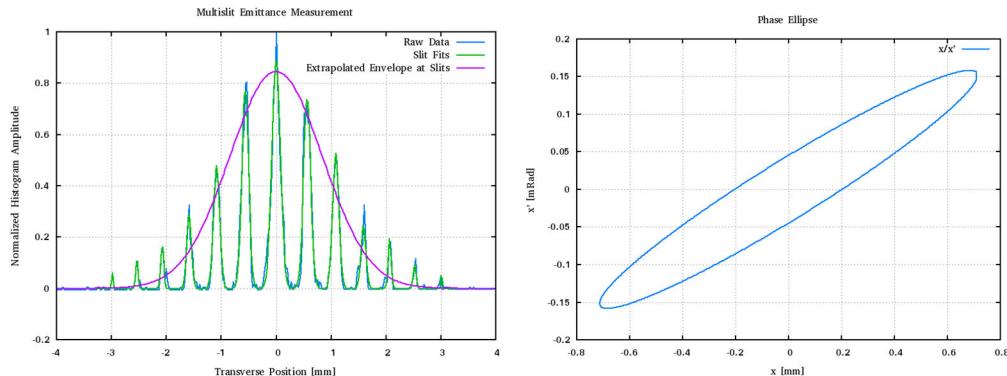


Fig 3 – A fit to the envelope (left) did not change the emittance when factored into the emittance calculation in normalizing each of the separate beamlet integrals. The phase ellipse from the Twiss parameters (right) has axis units of position (x [mm] along the x -axis) and divergence (x' [mrad] along the y -axis)

Appendix A – MyHistogram.java

```
import java.awt.Toolkit;
import java.awt.datatransfer.Clipboard;
import java.awt.datatransfer.StringSelection;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
public class MyHistogram {
    public static void main(String[] args) {
        BufferedImage img = null;
        try { img = ImageIO.read(new File("C:\\\\Temp\\\\sample.png"));
            } catch (IOException e) { e.printStackTrace(); }
        int myWidth; int myHeight;
        myWidth = img.getWidth();
        myHeight = img.getHeight();
        float mySum;
        int[][] myImg = new int[myWidth][myHeight];
        //float maxamp = 0;
        int i=0, j=0;
        String myOut = "";
        while (i < myWidth) {
            j=0;
            while ( j < myHeight ) {
                myImg[i][j] = img.getRGB(i, j);
                j++;
            }
            i++;
        }
        i=0;
        while ( i < myWidth){
            j = 0;
            mySum = 0;
            while (j < myHeight) {
                mySum += myImg[i][j];
                j++;
            }
            //mySum = mySum;
            myOut += mySum + "\n";
            i++;
        }
        StringSelection selection = new StringSelection(myOut);
        Toolkit defaultToolkit = Toolkit.getDefaultToolkit();
        Clipboard clp = defaultToolkit.getSystemClipboard();
        clp.setContents(selection, selection);
    }
}
```

Appendix B - fast_ms_emit.py

```

import matplotlib.pyplot as plt
from math import pi, isnan
from time import localtime, strftime, time as now
from numpy import arange, exp, abs, delete, float, min, max, array, inf, random, std, median
from scipy.optimize import leastsq as myFit
from scipy.integrate import quad as integrate
from subprocess import call
import tarfile
import commands
import sys
import os

t0 = now()
filedate = str( t0 )

# EXPERIMENT SETUP PARAMS
myW = 40e-6          # Slit Width [m] (d in paper)
myS = 400e-6          # Slit Spacing (between slits) [m] (w in paper)
myX107 = 9221.9455    # Survey distance along beamline to x107 [mm] (Slit position)
myX108 = 9676.6585    # Survey distance along beamline to x108 [mm]
myX109 = 10058.638    # Survey distance along beamline to x109 [mm]
myX111 = 10759.953    # Survey distance along beamline to x111 [mm]
myX115 = 14000.000    # Distance along beamline to x115 [mm] (Slit position) !!! not from survey and adds distance
through chicane !!!
myX120 = 16600.485    # Survey distance along beamline to x120 [mm]
myX124 = 19147.663    # Survey distance along beamline to x124 [mm]
z0 = 107
z1 = 111
z0 = myX107
z1 = myX111
myL = (z1 - z0)/1000 # 0.445           # Drift length between slit face and screen [m]
x_res = 9.3e-6         # X-Axis Resolution [m/px], (0.008 / 366) = 2.8758e-5 for initial sample

# CAMERA PARAMS
myCam = 4
myPlane = "X"
myAver = 5
MINIMUM_SLITS = 3

# TEST PARAMS
DEBUG = False
READ_LIVE = True
RANDOMIZE_NOISE = False
mySample = "/export/home1/edstrom/acl/emit_calc/sample.dat"
test_noise = 0.01
test_level = 0.0

# Paths
OUTPUT_DIRECTORY = "/usr/local/userb/fast/sync/" # Output Directory to be used
ARCHIVE_DIRECTORY = "/usr/local/cbs_files/cns_write/acl/data_files/fast_ms_emit/" # Archiving directory for tars
myHistogram = ARCHIVE_DIRECTORY + "fast_ms_emit.hist"
ACL = "/usr/local/xcons/bin/acl"
myHistFetch = "/usr/local/cbs_files/sequencer/acl/fast_gethist.acl"

# PROCESS PARAMS
smooth_range = 40          # Smoothing Range for Baseline Fit
pick_smooth = 4             # Smoothing factor for picking
min_peak_width = 20          # Minimum peak width for picking
min_threshold = 0.002        # Threshold over the standard background fit to start peak detection (0.1 = 10%)
n_pts = 0                   # Total Number of Points

# CONTROL FLAGS
SUBTRACT_BACKGROUND = 1     # 0 = no subtraction; 1 = fit subtraction; 2 = median subtraction
CENTER_ON_ENVELOPE = True   # Center all data around the envelope (rather than the central/highest peak)
VERBOSE = True               # Extra verbose output
FRAME_OUTPUT = False         # Only outputs the detected gaussians plus 10% (at most)
FULL_FIT = True              # Initially fit for the full ensemble of gaussians
FULL_OUTPUT = True           # Enable output to bd-www

y_in,x_in,y,x = [],[],[],[]      # Histogram functions, each with size of n_pts
a = []                           # Full fit parameter array, will be (n_bumps + 1) x 4 for the baseline and all other gaussians
p_full = []                       # A 1-D list of the a[] parameterization for the fit routine
                                  # Individual sets of gaussian fit parameters:
bump= []                          # 0 = Amplitude
i,j,k = 0,0,0                      # 1 = Centroid
myArg = []                         # 2 = Sigma
                                  # 3 = Linear offset
                                  # Individual bump information
                                  # Loop iterators
                                  # Command Line Arguments

#####
##### COMMAND LINE ARGUMENTS
print "Welcome to the FAST Multi-slit emittance scanner."

if (len(sys.argv) > 1):
    myArg = sys.argv[1].split(',')
    nArgs = len(myArg)
if (nArgs < 1):
    print "No command-line arguments passed..."
    print "> Syntax: python fast_ms_emit.py arg1=val1,arg2=val2..."
    print "> Valid arguments: <FILENAME>,cam=(1-4),plane(X/Y),aver=(int),"
    print ">           z0=(107/115),z1=(108/109/111/115/120/124),res=(float),"
    print ">           thresh=(float),minwidth=(int),bgssubtract=(0/1/2),"

```

```

print "> min_peaks=(int),plot=(0/1)"
print
print "Proceeding with default arguments..."
else:
    print "Parsing command-line arguments..."
i=0
while (i < nArgs):
    myArg[i] = myArg[i].split('=')
    if(myArg[i][0] == 'cam'):
        if((int(myArg[i][1]) > 0) and (int(myArg[i][1]) < 5)):
            print "> Using DC" + str(int(myArg[i][1])) + " for histogram information."
            myCam=int(myArg[i][1])
        else:
            print "> !! Invalid camera selection: [" + myArg[i][1] + "]"
    if(myArg[i][0] == 'res'):
        if(float(myArg[i][1]) > 0):
            print "> Using " + str(float(myArg[i][1])) + " as the camera resolution."
            x_res=float(myArg[i][1])
        else:
            print "> !! Invalid resolution: [" + myArg[i][1] + "]"
    if(myArg[i][0] == 'z0'):
        if (int(myArg[i][1]) == 107):
            l0 = 107
            z0 = myXl07
        if (int(myArg[i][1]) == 115):
            l0 = 115
            z0 = myXl15
        print "> Slits are presumed to be inserted at x" + str(l0) + "."
    if(myArg[i][0] == 'z1'):
        if (int(myArg[i][1]) == 108):
            l1 = 108
            z1 = myXl08
        if (int(myArg[i][1]) == 109):
            l1 = 109
            z1 = myXl09
        if (int(myArg[i][1]) == 111):
            l1 = 111
            z1 = myXl11
        if (int(myArg[i][1]) == 115):
            l1 = 115
            z1 = myXl15
        if (int(myArg[i][1]) == 120):
            l1 = 120
            z1 = myXl20
        if (int(myArg[i][1]) == 124):
            l1 = 124
            z1 = myXl24
        print "> YAG screen and camera are presumed to be active at x" + str(l0) + "."
    if(myArg[i][0] == 'min_peaks'):
        if (int(myArg[i][1]) > 2):
            print "> Using " + str(int(myArg[i][1])) + " as the minimum number of peaks to accept for fitting."
            MINIMUM_SLITS=int(myArg[i][1])
        else:
            print "> !! Invalid peak number: [" + myArg[i][1] + "]"
    if(myArg[i][0] == 'plot'):
        if (int(myArg[i][1]) == 0):
            print "> Not displaying process plots."
            DEBUG = False
        else:
            print "> Displaying process plots... close each plot to continue."
            DEBUG = True
    if(myArg[i][0] == 'bgsubtract'):
        if (int(myArg[i][1]) == 1):
            print "> Subtracting out the fit background."
            SUBTRACT_BACKGROUND = 1
        elif (int(myArg[i][1]) == 2):
            print "> Subtracting out the median value (no fit to the background)"
            SUBTRACT_BACKGROUND = 2
        else:
            print "> Not subtracting out the measured background."
            SUBTRACT_BACKGROUND = 0
    if(myArg[i][0] == 'thresh'):
        if(float(myArg[i][1]) > 0):
            print "> Using " + str(float(myArg[i][1])) + " as the minimum fit threshold (for peak determination)."
            min_threshold=float(myArg[i][1])
        else:
            print "> !! Invalid minimum fit threshold: [" + myArg[i][1] + "]"
    if(myArg[i][0] == 'minwidth'):
        if(int(myArg[i][1]) > 0):
            print "> Using " + str(int(myArg[i][1])) + " as the minimum peak width [px] (for peak determination)."
            min_peak_width=int(myArg[i][1])
        else:
            print "> !! Invalid minimum peak width in px: [" + myArg[i][1] + "]"
    if(myArg[i][0] == 'aver'):
        if(int(myArg[i][1]) > -1):
            print "> Averaging " + str(int(myArg[i][1])) + " histogram readbacks for measurement."
            myAver=int(myArg[i][1])
        else:
            print "> !! Invalid averaging request: [" + myArg[i][1] + "]"
    if(myArg[i][0] == 'plane'):
        if ((myArg[i][1] == 'X') or (myArg[i][1] == 'x') or (myArg[i][1] == 'Y') or (myArg[i][1] == 'y')):
            print "> " + myArg[i][1] + "-axis histogram to be used for collection."
            myPlane = myArg[i][1]
        else:
            print "> !! Invalid plane selection: [" + myArg[i][1] + "]"

```

```

if(myArg[i][0].find(".tar") > 0):
    READ_LIVE = False
    print "> tar source: " + myArg[i][0]
    tar = tarfile.open(myArg[i][0], 'r')
    tar.extractall()
    tar.close()
    mySample = "ms emit.hist"
if(myArg[i][0].find(".dat") > 0) or (myArg[i][0].find(".hist") > 0):
    READ_LIVE = False
    print "> data source: " + myArg[i][0]
    mySample = myArg[i][0]
i+=1
myL = (z1 - z0)/1000
print
#quit()
#####
##### UTILITY FUNCITONS
def make_float_array(a_in):
    a_out = [0.0] * len(a_in)
    i=0
    while ( i < len(a_in) ):
        a_out[i] = float(a_in[i])
        i+=1
    return a_out

def make_random_array(a_pts):
    a_out = [0.0] * len(a_in)
    for i in a_out:
        i = random.random()
    return a_out

def smooth_array(a_in, sf):
    a_out = [0.0] * len(a_in)
    i = 0
    while ( i < len(a_in)):
        j=i-sf
        k=i+sf
        if (j < 0): j=0
        if (k > len(a_in)): k = len(a_in)
        a_out[i] = (sum(a_in[j:k])/(k-j))
        i+=1
    return a_out

def form_array(a_in, n_d):
    a_out = []
    i=0
    while ( i < (len(a_in) / n_d)):
        a_part = []
        j=0
        while ( j < n_d ):
            a_part.append(a_in[(i*n_d)+j])
            j+=1
        a_out.append(a_part)
        i+=1
    return a_out

def flatten_array(a_in):
    a_out = []
    for i in a_in:
        for j in i: a_out.append(j)
    return a_out

def plot_arrays(Y,X):
    print sat() + " -- Close the current plot to continue...""
    plt.figure()
    i=0
    if (len(X) < len(Y)):
        while(i < len(Y)):
            plt.plot(X[0],Y[i])
            i=i+1
    else:
        while(i < len(Y)):
            plt.plot(X[i],Y[i])
            i=i+1
    plt.get_current_fig_manager().window.set_position(1)
    plt.show()
    return 0

def find_array_index(a_in, s):
    b_in = arange(0, len(a_in))
    i=0
    while ( i < len(a_in)):
        if ( a_in[i] == s ): return b_in[i]
        i+=1
    return None

def find_FWHM(a_in, min_pts, thresh):
    a_max = find_array_index(a_in, max(a_in))
    i = a_max
    j = a_max
    while ( i > 0 ):
        if ( a_in[i] < ((a_in[a_max] + thresh)/2)): break
        i=i-1
    while ( j < len(a_in) ):
        if ( a_in[j] < ((a_in[a_max] + thresh)/2)): break
        j=j+1

```

```

if ((j-i) < min_pts):
    i=0
    j=len(a_in)
    return [i, j]

def find_peaks(a_in, min_gap, thresh):
    a_out=[]
    peak_det = False
    a_pts = len(a_in)
    a_mod = smooth_array(a_in, (pick_smooth))
    i, j, k, n, myPeakIndex, myCent, myPeak, maxPeak = 0, 0, 0, 0, 0, 0, 0.0, 0.0
    while (i < len(a_mod)):
        if ( a_mod[i] > thresh) and (not peak_det):
            if ( DEBUG ):
                print("Begin: " + str(i) + ":" + str(i*x_res) + ", " + str(a_mod[i]))
            peak_det = True
            j = i
            i += (min_gap/2)
            if (i >= a_pts): i = a_pts - 1
        if ( (a_mod[i] < thresh) or (i == a_pts - 1) ) and (peak_det):
            if ( DEBUG ):
                print("End: " + str(i) + ":" + str(i*x_res) + ", " + str(a_mod[i]))
            print
            peak_det = False
            k = i
            i += (min_gap/2)
            a_eval = a_in[j:k]
            myPeak = max(a_eval)
            myCent = (find_array_index(a_eval, max(a_eval))+j)
            a_out.append( [myPeak, myCent] )
            if (myPeak > maxPeak):
                maxPeak, myPeakIndex = myPeak, n
            n+= 1
            i+=1
        i=0
    while (i < len(a_out)): # FIND PEAK WINDOWS
        myCent = a_out[i][1]
        if (len(a_out) <= 1): # Catch if only a single peak is detected above
            return [0],0
        if (i == 0):
            myWid = (a_out[i+1][1] - myCent)
        else:
            myWid = (myCent - a_out[i-1][1])
        j = myCent - (myWid/2)
        if (j < 0): j=0
        k = myCent + (myWid/2)
        if (k >= len(a_in)): k = (len(a_in)-1)
        #      Full Width x0      x1
        a_out[i] = a_out[i] + [myWid] + [j] + [k]
        i+=1
    return a_out, myPeakIndex

def snow(): return strftime("%H:%M:%S %m-%d-%Y", localtime())
def snowfile(): return strftime("%Y-%m-%d %H-%M-%S", localtime())
def sfiletime(): return strftime("%H:%M:%S %m-%d-%Y", localtime(myFileStat.st_ctime))
def sdt(): return ('{:>6.2f}'.format(now() - t0))
def f2s(a_in,b):
    a_out = '{0: .' + str(b) + 'f}'
    return a_out.format(a_in)
def e2s(a_in,b):
    a_out = '{0: .' + str(b) + 'e}'
    return a_out.format(a_in)
def i2s(a_in,b):
    a_out = '{0: .' + str(b) + 'd}'
    return a_out.format(a_in)

def get_histogram(c, myPlane, nAver): # c = camera = 1-4, myPlane = "X" or "Y", nAver >= 1
    myACLString = ACL + " " + myHistFetch + " N:DC" + str(c) + myPlane + "H " + str(nAver) + " > " + myHistogram
    os.system(myACLString)
    return

def set_output(b):
    myACLString = ACL + " \\"enable settings;" 
    i = 0
    while (i < len(b)):
        myACLString += "set N:MSEMIT[" + str(i) + "] "
        if (not isnan(b[i])):
            myACLString += str(float(b[i])) + ";"
        else:
            myACLString += "-999;"
        i+=1
    myACLString += "\\""
    result=commands.getstatusoutput(myACLString)
    return result

#####
##### FIT FUNCTIONS
def peval_full(q, b):
    # This is a full evaluation of the gaussians without a baseline... this ignores the baseline fit in a[n_bumps]
    ptot = [0.0]*len(q)
    i=0
    while (i < n_bumps):
        if ( len(b) > n_bumps ):
            p_test = [b[i*4],b[(i*4)+1],b[(i*4)+2],b[(i*4)+3]] # For if b is a [1 x (4*n_bumps)] array
        else:
            p_test = b[i] # For if b is a [4 x n_bumps] array

```

```

ptot = ptot + peval(q, p_test) # Single gaussian test function
i=i+1
return ptot

def pevalb_full(q, b):
    # This is a full evaluation of the gaussians including the baseline (in a[n_bumps])
    ptot = [0.0]*len(q)
    i=0
    while (i < n_bumps):
        if ( len(b) > n_bumps ):
            p_test = [b[i*4],b[(i*4)+1],b[(i*4)+2],b[(i*4)+3]] # For if b is a [1 x (4*n_bumps)] array
        else:
            p_test = b[i] # For if b is a [4 x n_bumps] array
        ptot = ptot + peval(q, p_test) # Single gaussian test function
        i=i+1
    if ( len(b) > n_bumps ):
        p_test = [b[i*4],b[(i*4)+1],b[(i*4)+2],b[(i*4)+3]]
    else:
        p_test = b[i]
    ptot = ptot + pevalb(q, p_test) # Baseline test function
    return ptot

def pevalb(q, b): return ((b[0]*exp(-0.5*((q-b[1])/b[2])**2))) + b[3] # A gaussian on a linear offset (to be used for the baseline)
def peval(q, b): return (b[0]*exp(-0.5*((q-b[1])/b[2])**2)) # A gaussian with no linear offset (to be used for each of the superimposed gaussians)
def pfunc(b, r, q): return (r - peval(q, b))
def pfuncb(b, r, q): return (r - pevalb(q, b))
def pfunct(b, r, q): return (r - peval_full(q, b))
def pfunctfb(b, r, q): return (r - pevalb_full(q, b))
def fit(function, p_ls, y_ls, x_ls):
    errfunc = lambda p_ls, x_lsl, y_lsl: function(p_ls, y_lsl, x_lsl)
    pfit, pcov, infodict, errmsg, success = myFit(errfunc, p_ls, args=(x_ls, y_ls), full_output=1, epsfcn=0.0001)
    chisq = ((infodict['fvec'])**2).sum()/(len(y_ls) - len(p_ls))
    # chisq is the same as (errfunc(pfit, x_ls, y_ls)**2).sum()/(len(y_ls)-len(p_ls))
    if (len(y_ls) > len(p_ls)) and pcov is not None:
        pcov = pcov * chisq
    else:
        pcov = inf
    pfit_out = [0.0]*len(pfit)
    perr_out = [0.0]*len(pfit)
    i=0
    while (i < len(pfit)):
        pfit_out[i] = pfit[i] # format the output array into a standard list
        try: perr_out[i] = (abs(pcov[i][i])**0.5)
        except: perr_out[i] = 0.0
        i=i+1
    return [pfit_out, perr_out, pcov, chisq]

#####
##### ELLIPSE EVALUATION FUNCTIONS
# The phase ellipse is based on twiss parameters: Alpha, Beta, Gamma, Epsilon
# (Epsilon = emittance)
# x' = (B +/- sqrt(B^2 - 4AC)) / 2A
#   A = Beta
#   B = -2 * Alpha * x
#   C = Gamma * x^2 - Epsilon
# x limits @ ( B^2 - (4 * A * C) ) = 0
# -> x = +/-((myBeta * myEmitt)/(myBeta * myGamma - myAlpha^2))^0.5
def ellipse_limits(b):
    q = [abs((b[1] * b[3])/(b[1] * b[2] - b[1]**2))**0.5, abs(-1*((b[1] * b[3])/(b[1] * b[2] - b[1]**2))**0.5)]
    return (1.25*max(q)) # A little padding

def eval_ellipse(b, n, q_lim): # but I'll just use the q_lim=secMom[0]
    a_out = []
    b_out = []
    #q_lim = ellipse_limits(b)
    if (not isnan(q_lim)):
        q = arange(-1*q_lim,q_lim,((2*q_lim)/n))
        f = eval_ellipse_pos(q, b)
        g = eval_ellipse_neg(q, b)
        i=0
        while (i < len(f)):
            if (not isnan(f[i])):
                a_out.append(q[i])
                b_out.append(f[i])
            i=i+1
        i = len(g) - 1
        while (i > 0):
            if (not isnan(g[i])):
                a_out.append(q[i])
                b_out.append(g[i])
            i=i-1
        a_out.append(a_out[0]) # Close the ellipse
        b_out.append(b_out[0])
    else:
        print "      -- WARNING: Cannot create ellipse with the given parameters:"
        print "      p_ellipse: " + str(b)
        print "      secMom[1]: " + str(q_lim)
        a_out.append(0.0)
        b_out.append(0.0)
    return a_out, b_out

def eval_ellipse_pos(q, b): # Positive half of the sqrt
    # ( ( B^2 - 4 * A * C )^0.5 )/(2 * A )
    return ( ( (-2*b[0]*q) + ( (-2*b[0]*q)**2 - (4 * b[1]* ((b[2]*(q**2)) - b[3])) )**0.5 )/(2 * b[1]))
```

```

def eval_ellipse_neg(q, b): # Negative half of the sqrt
    #   ( ( - B^2 - ( (-2*b[0]*q) - ( (-2*b[0]*q)**2 - (4 * b[1]* ((b[2]*(q**2)) - b[3])) ) )**0.5 )/(2 * A )
    return ( ( (-2*b[0]*q) - ( (-2*b[0]*q)**2 - (4 * b[1]* ((b[2]*(q**2)) - b[3])) ) )**0.5 )/(2 * b[1]))

#####
##### READ IN DATA AND FORMAT IT #####
if (READ_LIVE):
    print sdt() + " -- Reading data from the DC" + str(myCam) + " " + myPlane + "-Axis Histogram (" + str(myAver) + "x)"
    print "(" + myHistogram + ")"
    get_histogram(myCam,myPlane,myAver) # e.g. Camera #3, Y-Axis, 1 averages...
    myFile = open(myHistogram,"r")
else:
    print sdt() + " -- Reading data from " + mySample
    myFileStat = os.stat(mySample)
    myFile = open(mySample,"r")           # Get File info
y_in = myFile.read().split('\n')          # Read in data
myFile.close()
for myLine in y_in:
    myPart = myLine.split('\t')
    if(len(myPart) > 4):
        try:
            y.append(float(myPart[4]))
            x.append(float(myPart[2]))
        except:
            pass
    else:
        try:
            y.append(float(myPart[0]))
        except:
            pass
n_pts = len(y)
if(n_pts == 0):
    print
    print "!! Failed to parse input data file !!"
    quit()
if (True):
    i = n_pts - 1
    while i > 0:
        if y[i] == 0:
            y = delete(y,i)           # Remove all trailing 0's
            n_pts-=1
            i-=1
        else:
            i=0
print sdt() + " Number of points in dataset: " + str(n_pts)
if(n_pts < 100):
    print " Too few points! (Parsed=" + str(n_pts) + " / File=" + str(n_pts_ini) + ")"
    quit()
if(len(x) == n_pts):
    print " -> Using x-axis information in units of [px]"
else:
    print " -> Inconsistent or non-existent x-axis information available"
    x = arange(0.0,n_pts,1.0)
    x *= x_res

print " Slits @ x" + str(l0) + ", YAG Screen @ x" + str(l1) + ", Total distance: " + str(myL) + "m."
print

if (DEBUG): plot_arrays([y],[x])

#####
##### ADD NOISE & NOISE FLOOR TO TEST DATA #####
if (not READ_LIVE) and (RANDOMIZE_NOISE):
    test_noise = test_noise * random.random()
    test_level = test_level * random.random()
    i=0
    print " -- Adding random noise (" + test_noise + ") and level (" + test_level + ") for testing..."
    while(i < n_pts):
        y[i] += (((random.random()-0.5)*2*test_noise) + test_level)
        #if ( y[i] < 0.0 ): y[i] = 0
        i+=1

#####
##### FIND PEAKS #####
if (DEBUG):
    i = 0
    while (( i < 10 ) and ( i < n_pts)):
        if (i == 0):
            print "      y: [" + i2s(i,3) + "] " + e2s(y[i],4)
        else:
            print "      [" + i2s(i,3) + "] " + e2s(y[i],4)
        i+=1
    if (i < n_pts): print "..."

y_s = smooth_array(y,smooth_range)
ps = [(max(y_s) - min(y_s)), ((max(x) - min(x))/2), ((max(x) - min(x))/20), min(y_s)]
#if (DEBUG): plot_arrays([y_s,y],[x])
ps,psc,psc_s = fit(pfucb, ps, y_s, x) # Fit to the smoothed data to find a baseline fit
lin_baseline = ps[3]
print sdt() + " -- Linear Baseline = " + f2s(lin_baseline,4)
noise_level = abs(min(y - lin_baseline))
print sdt() + " -- Noise Level = " + e2s(noise_level,4)
pick_threshold = (lin_baseline) + ((max(y) - min(y))*min_threshold) + (noise_level/2)
print sdt() + " -- Peak Pick Thresh = " + e2s(pick_threshold,4)

if (DEBUG): # Check smooth fit to linear baseline

```

```

y_s_eval = pevalb(x,ps)
plot_arrays([y,y_s,y_s_eval],[x])

detect_threshold = 1.0 - ((noise_level/max(y))/0.35)
if (detect_threshold < 0.0): detect_threshold = 0.0
print sdt() + " -- Peak Detection = " + f2s(detect_threshold,2)
if (detect_threshold > 1):
    print sdt() + " -- No Discernible Peaks For Fitting!! Quitting..."
    quit()

bump, x_cent = find_peaks(y, min_peak_width, pick_threshold)
n_bumps = len(bump)
if (n_bumps < MINIMUM_SLITS):
    print sdt() + " -- Detected only " + str(n_bumps) + " peak(s)! Quitting..."
    quit()
print sdt() + " -- Number of peaks found: " + str(n_bumps)
print sdt() + " -- Center index: " + str(x_cent)

# Find the bump data above the FWHM and the FWMF data for each
x_fwhm,y_fwhm,x_part,y_part=[[],[],[],[]]
i = 0
while (i < n_bumps):
    y_part.append(y[bump[i][3]:bump[i][4]])
    x_part.append(x[bump[i][3]:bump[i][4]])
    i+=1

if (DEBUG): plot_arrays(y_part,x_part)

#####
##### PERFORM INITIAL (SINGLE-GAUSSIAN) FIT(S)
# Set initial guesses for fitting
print sdt() + " -- Fitting single gaussians"
a = ([[0.0, 0.0, 1.0, 0.0]]) * (n_bumps + 1)
ae=[[0.0, 0.0, 0.0, 0.0]] * (n_bumps + 1)
full_chisq = 0.0
i=0
while (i < n_bumps): # Find Initial Fit Guesses for each bump
    # Amp Cent Sigma Lin Off
    a[i] = ([bump[i][0], x[bump[i][1]], (bump[i][2] * x_res)/8, lin_baseline])
    i+=1
i=0
while(i < n_bumps):
    a[i], ae[i], p_conv, p_chisq = fit(pfuncb, a[i], y_part[i], x_part[i]) # Perform Fit (including baseline)
    full_chisq += (p_chisq / n_bumps)
    i+=1

#####
##### FIND A BASELINE FIT / MEDIAN
lin_offset = median(y) # Also used as a fit parameter for the envelope if no background subtraction is performed
if (SUBTRACT_BACKGROUND == 1):
    print sdt() + " -- Fitting and subtracting the baseline"
    y_residual = (y - pevalb_full(x, flatten_array(a)))
    ys_res = smooth.array(y_residual,smooth_range)
    pb=[(max(ys_res)-min(ys_res))/2, (max(x)-min(x))/2, (max(x)-min(x))*10, lin_baseline)]
    pb,pbe,pbc,pbcs = fit(pfuncb, pb, ys_res, x) # Perform smoothed fit (including baseline)
    pb,pbe,pbc,pbcs = fit(pfuncb, pb, y_residual, x) # Perform baseline fit (including baseline)
    y_base = pevalb(x, pb) # Adjust these functions as needed to fit to the background
    if (DEBUG):
        plot_arrays([y_residual,y_base],[x])
    a[n_bumps] = pb
    y -= y_base
    y_s -= y_base
elif (SUBTRACT_BACKGROUND == 2):
    print sdt() + " -- Subtracting the median"
    a[n_bumps] = [0,0,0,lin_offset]
    y_base = []
    i = 0
    while(i < n_pts):
        y_base.append(a[n_bumps][3])
        i += 1
    if (DEBUG):
        plot_arrays([y,y_base],[x])
    y -= a[n_bumps][3]
    y_s -= a[n_bumps][3]

i=0
while(i <= n_bumps): # Remove the baseline offset on all individual gaussians...
    a[i][3] = 0.0
    i+=1

#####
##### PERFORM FULL FIT (IF DESIRED)
if FULL_FIT:
    print sdt() + " -- Performing Full Fit"
    p_full, pe_full, full_cov, full_chisq = fit(pfuncfb, flatten_array(a), y, x) # Perform Fit
    a = form_array(p_full, 4)
    print sdt() + " -- Full Fit Chi^2 = " + e2s(full_chisq,4)

#####
##### RE-FIT/ELIMINATE BAD GAUSSIANS
print sdt() + " -- Eliminating Bad Gaussians... "
# Sigma sanity check on background
if ((not SUBTRACT_BACKGROUND) and (abs(a[n_bumps][2]) < 1e-4)):
    print "    --> Bad Background Fit (Params: " + str(a[i]) + ")"
    a[n_bumps][0] = 0.0
    a[n_bumps][3] = lin_baseline
    p_full, pe_full, full_cov, full_chisq = fit(pfuncfb, flatten_array(a), y, x) # Perform Fit

```

```

i=0
while (i < n_bumps):
    # Amplitude and Sigma sanity checks on individual gaussians
    if (a[i][0] < 0.0 or ((abs(a[i][2]) > 1e-3) or (abs(a[i][2]) < 1e-7))):
        print "    " + str(i+1) + " --> Bad Fit (Params: " + str(a[i]) + "... ignoring in calculations"
        a[i][0] = 0.0
        a[i][2] = 1.0
    i+=1

##### INTEGRATE UNDER BEAMLET GAUSSIANS
i=0
I=[]
p_eval=[0.0, 0.0, 0.0, 0.0]
while (i < n_bumps):
    p_eval = [(a[i][0] - a[i][3]), a[i][1], a[i][2], 0.0] # Is this fair to do if we don't do the background subtraction?
    Otherwise we just want p_eval = a[i]
    result = integrate(peval, min(x), max(x), p_eval)
    I.append(result[0]) # Integration for each gaussian
    i+=1

##### FIT TO ENVELOPE AT THE SLIT MASK
print sdt() + " -- Fitting to the envelope at the slit mask"
x_env,y_env=[[],[]]
i=0
while (i<n_bumps):
    if (a[i][0] > 0.0 and I[i] > 0.0): # Ignore bad gaussians marked above
        m = (i-x_cent)
        x_env.append((m*myS) + a[x_cent][1]) # At the slit mask all points will by definition be at intervals of myS
        #y_env.append(a[i][0]) # Use max amplitude for each fit beamlet gaussian
        y_env.append(max(y)*I[i]/max(I)) # Use a scaled integral for each beamlet gaussian
    i+=1
if(SUBTRACT_BACKGROUND > 0):
    p_env = [(max(y_env)), (max(x_env) + min(x_env))/2, max(x_env)/2, 0.0]
    p_env, pe_env, env_cov, env_chisq = fit(pfunc, p_env, y_env, x_env) # Perform Fit
    y_env_eval = peval(x, p_env)
else:
    p_env = [(max(y_env)), (max(x_env) + min(x_env))/2, max(x_env)/2, lin_offset]
    p_env, pe_env, env_cov, env_chisq = fit(pfncb, p_env, y_env, x_env) # Perform Fit with baseline
    y_env_eval = pevalb(x, p_env)
    i = 0
    while (i < n_pts):
        if (y_env_eval[i] < 0.0):
            y_env_eval[i] = 0.0
        i += 1
print sdt() + " -- Envelope Fit Chi^2 = " + e2s(env_chisq,4)
print sdt() + " -- Envelope Centroid = " + e2s(p_env[1],4)

if (DEBUG):
    plot_arrays([y,y_env_eval,y_env],[x,x,x_env])

##### INTEGRATE UNDER BACKGROUND/ENV GAUSSIANS
result = integrate(pevalb, min(x), max(x), args=(a[n_bumps]))
I.append(result[0]) # Background integration in I[n_bumps]

result = integrate(peval, min(x), max(x), args=(p_env)) # Evaluate the envelope with no baseline
I.append(result[0]) # Envelope integration in I[n_bumps + 1]

##### CENTER ON ENVELOPE CENTROID
print sdt() + " -- Centering on the Envelope Centroid..."

if (CENTER_ON_ENVELOPE):
    trans_offset = p_env[1] # Remove the offset between original x and envelope centroid
    i=n_bumps
    i_offset = 0
    while (i >= 0):
        if ( (a[i][1] - trans_offset) > 0 ): i_offset = i # Find the peak just above the envelope centroid... aka a good solid
peak
        i-=1
    if (i_offset > 0):
        frac_offset = (a[i_offset][1] - trans_offset) / (a[i_offset][1] - a[i_offset-1][1])
    else:
        CENTER_ON_ENVELOPE = False
        trans_offset = a[x_cent][1] # Just center on the peak after all...
        frac_offset = 0.0
else:
    trans_offset = a[x_cent][1] # Remove the offset between original x and highest peak

# Translate x-axis components
x -= trans_offset

# Translate all fit parameters
x_min, x_max = x[len(x)-1],0.0
i=0
while (i < n_bumps):
    a[i][1] -= trans_offset
    if ( a[i][1] > x_max ): x_max = a[i][1]
    if ( a[i][1] < x_min ): x_min = a[i][1]
    i+=1
a[n_bumps][1] -= trans_offset # Shift Background as well
p_env[1] -= trans_offset

print "    -- Centering on Envelope: " + str(CENTER_ON_ENVELOPE)
print "    -- Positive Center Index: " + str(i_offset)
print "    -- Fractional Offset: " + e2s(frac_offset,4)

```

```

#####
##### FIND EMITTANCE PARAMETERS #####
i=0
if CENTER_ON_ENVELOPE:
    myOffset = (i_offset - frac_offset) # Center about Envelope
else:
    myOffset = x_cent # Center about highest peak

myDiv = []
secMom = [0.0] * 3
myIntSum = 0.0
while ( i < n_bumps ):
    if (a[i][0] > 0.0): # So long as there's a peak, include it in the calculation...
        m = (i-myOffset)
        myIntSum = myIntSum + I[i] # Sum Integrals
        myDiv.append((a[i][1] - (m * mys)) / (myL)) # Calculate Divergence
    else:
        myDiv.append(0.0)
    i+=1
i=0
while ( i < n_bumps ):
    if (a[i][0] > 0.0):
        m = (i-myOffset)
        secMom[0] = secMom[0] + (I[i] * (m*mys)**2) # <x^2>
        secMom[1] = secMom[1] + (I[i] * ((myDiv[i])**2 + ((a[i][2])**2 - ((myW**2)/12.0))/(myL**2))) # <x'^2>
        secMom[2] = secMom[2] + (I[i] * (m*mys) * myDiv[i]) # <xx'>
    i+=1
i=0
while ( i < len(secMom)):
    secMom[i] = (secMom[i] / (myIntSum))
    i+=1
myEmitSq = (secMom[0]*secMom[1]) - (secMom[2]*secMom[2])
myEmit = (myEmitSq)**(0.5)
mydEmit = (p_env[2] * secMom[1]) / myEmit # Gives differential as a fraction of the Emittance
myEmitum = myEmit * 1000.0 * 1000.0 # x(mm/m) & x'(mrad/rad) = um
mydEmitum = mydEmit * myEmitum # Statistical Error
myAlpha = -1.0 * (secMom[2]/myEmit)
myBeta = secMom[0]/myEmit
myGamma = (1.0 + (myAlpha**2))/myBeta

print sdt() + " -- Emittance = " + f2s(myEmitum,4) + " +/- " + e2s(mydEmitum,4) + " um (" + e2s(myEmit,4) + " m*Rad)"
print " Emit = sqrt(" + e2s(secMom[0]*secMom[1],4) + " - " + e2s((secMom[2]*secMom[2]),4) + " ) m*Rad"

#####
##### CONTOUR PLOT #####
#####
##### FRAME OUTPUT AND GENERATE CURVES #####
xi_min = 0
xi_max = (len(x) - 1)
if (FRAME_OUTPUT):
    x_margin = (x_max - x_min) * 0.15
    x_min -= x_margin
    x_max += x_margin
    while (x[xi_min] < x_min): xi_min += 1
    while (x[xi_max] > x_max): xi_max -= 1
x_out = x[xi_min:xi_max]
y_out = y[xi_min:xi_max]
y_fit = pevalb_full(x_out, flatten_array(a))
y_env = pevalb(x_out,p_env)
y_s_out = y_s[xi_min:xi_max]

#####
##### CALCULATE PHASE ELLIPSE #####
if (not isnan(myEmit)):
    p_ellipse = [myAlpha, myBeta, myGamma, myEmit]
    x_ellipse, y_ellipse = eval_ellipse(p_ellipse, len(x), (secMom[0]**(0.5)))
else:
    print " WARNING! -> Non-physical parameters found!"
    x_ellipse = [0.0] * n_pts
    y_ellipse = [0.0] * n_pts

#####
##### OUTPUT DATA #####
if (VERBOSE):
    print
    print " -- SUMMARY --"
    print "     Second Moments:"
    print "         <x^2> = " + e2s(secMom[0],4)
    print "         <x'^2> = " + e2s(secMom[1],4)
    print "         <xx'> = " + e2s(secMom[2],4)
    print
    print "     Alpha : " + f2s(myAlpha,4)
    print "     Beta : " + f2s(myBeta,4)
    print "     Gamma : " + f2s(myGamma,4)
    print
    print "     Phase Ellipse:"
    print "         Max x : " + e2s(max(abs(x_ellipse)), 4)
    print "         Max x' : " + e2s(max(abs(y_ellipse)), 4)
    print
    print " -- Bump Detection"
    print "     Bump YMax X(YMax) Width Xi Xf"
k=1
for i in bump:
    myOut = i2s(k,3)
    for j in i:

```

```

myOut += " " + e2s(j,3)
print myOut
k+=1
print
print " -- Final Fit Params & Div Calc"
print " Bump      Amp      Cent      Sig      LinOff     Integral    Divergence"
k=1
for i in a:
    myOut = i2s(k,3)
    for j in i:
        myOut += " " + e2s(j,3)
    myOut += " " + e2s(I[k-1],3)
    if (k > n_bumps):
        print
        print "  Background fit:"
    else:
        myOut += " " + e2s(myDiv[k-1],3)
    print myOut
    k+=1
print

if (FULL_OUTPUT):
    myOut = ""
    print sdt() + " -- Saving Summary to " + OUTPUT_DIRECTORY + "ms_emit.htm"
    myOut += "<html><body><div><span style='float:left;font-size:20px;'>Multislit Emittance Summary</span><pre style='float:right;'>" + snow() + "</pre><br>"
    if (not READ_LIVE):
        myOut += "<pre>File: " + mySample + " (" + sfiletime() + ")</pre>"
    myOut += "<table width=100%><tr>"
    myOut += "<td valign=top><div><pre>\n<b>Emittance & System Params:</b>\n\n&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbspEmittance (Un-Norm) : <b></b>" + e2s(myEmitun,3) + " &mu;m</b>\n\n&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbspDist to Screen (L) : " + e2s(myL,3) + " m\n\n&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbspSlit Width (w) : " + e2s(myW,3) + " m\n\n&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbspSlit Spacing (s) : " + e2s(myS,3) + " m\n\n&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbspHistogram Amplitude : " + f2s(max(y),2) + " ArbU\n\n&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbspHistogram Width : " + str(n_pts) + " px\n\n&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbspCamera Resolution : " + e2s(x_res,3) + " m/px\n</pre></div></td>\n\n"
    myOut += "<td valign=top><div><pre>\n<b>Second Moments:</b>\n\n&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&lt;x^2> : " + e2s(secMom[0],3) + " m<sup>2</sup>\n\n&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&lt;x^2gt; : " + e2s(secMom[1],3) + " rad<sup>2</sup>\n\n&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&lt;xx^gt; : " + e2s(secMom[2],3) + " m*rad\n<b>Checks:</b>\n\n&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&lt;full_chisa> : " + e2s(full_chisa,3) + "\n\n&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&lt;&sigma;<sub>Env</sub><sup>2</sup> : " + e2s((p_env[2]**2),3) + " m<sup>2</sup>\n\n&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&lt;x^2gt / &sigma;<sub>Env</sub><sup>2</sup> : " + f2s((secMom[0]/(p_env[2]**2)),3) + "\n</pre></div></td>\n\n"
    myOut += "<td valign=top><div><pre>\n<b>Twiss Parameters:</b>\n\n&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&alpha; : " + f2s(myAlpha,3) + "\n\n&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&beta; : " + f2s(myBeta,3) + "\n\n&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&gamma; : " + f2s(myGamma,3) + "\n\n"
    myOut += " <b>Background Subtraction:</b>\n      &ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&nbs
    if (SUBTRACT_BACKGROUND == 1):
        myOut += "Gaussian Fit"
    elif (SUBTRACT_BACKGROUND == 2):
        myOut += "Median"
    else:
        myOut += "None"
    myOut += "\n"
    if (FULL_FIT):
        myOut += " <b>Full Final Fit:</b>\n      &ampnbsp&ampnbsp&ampnbsp&ampnbsp&ampnbsp&nbsp;--> " + str(len(p_full)) + " Parameters"
    else:
        myOut += " <b>Single Gaussian Fits Only</b>\n
    myOut += "</pre></div></td>\n</tr></table>\n
    myOut += "<table width=100%><tr><td width=50%><img width=100% src='./ms_emit.png'></td>\n<td><img width=100% src='./ms_phase.png'></td></tr><tr><td colspan=2><center>\n<div><pre>" + str(n_bumps) + " Gaussians Detected (Threshold = " + f2s(pick_threshold,3) + " | Noise = " + f2s(noise_level,3)
+ "</pre><hr><pre><table width=100%>\n<tr><td>Gaussian</td><td>Amplitude</td><td>Centroid [m]</td><td>Sigma [m]</td><td>LinOff</td><td>Integral [m]</td><td>Div [Rad]</td></tr>\n    i=0\n    while(i < n_bumps):\n        myOut += "<tr><td>" + str(i+1) + "</td><td>" + e2s(a[i][0],3) + "</td><td>" + e2s(a[i][1],3) + "</td><td>" +
e2s(abs(a[i][2]),3) + "</td><td>--</td><td>" + e2s(I[i],3) + "</td><td>" + e2s(myDiv[i],3) + "</td><tr>" + i+1
        i = n_bumps
    if(SUBTRACT_BACKGROUND == 1):
        myOut += "<tr><td>Baseline</td><td>" + e2s(a[i][0],3) + "</td><td>" + e2s(a[i][1],3) + "</td><td>" + e2s(abs(a[i][2]),3) +
"</td><td>" + e2s(a[i][3],3) + "</td><td>" + e2s(I[i],3) + "</td><td>" + e2s(p_env[0],3) + "</td><td>" + e2s(p_env[1],3) + "</td><td>" + e2s(abs(p_env[2],3) +
"</td><td>" + e2s(lin_offset,3) + "</td><td>" + e2s(I[i+1],3) + "</td><td>" + e2s(myDiv[i+1],3) + "</td><tr>" + i+1
    if(SUBTRACT_BACKGROUND == 2):
        myOut += "<span style='float:left'>Median Baseline (subtracted) : " + e2s(lin_offset,3) + "</span><br>" + "
```

```

myFile.write(myOut)
myFile.close()

myOut = ""
print sdt() + " -- Saving Plot Data to " + OUTPUT_DIRECTORY + "ms_emit.dat"
myOut = "x x\' x [px] x [m] y y_fit y_envelope\n"
i=0
j=0
while ((i < len(x_ellipse)) or (i < len(x_out))): 
    if ( i < len(x_ellipse) ):
        myOut += str(x_ellipse[i]) + "\t" + str(y_ellipse[i])
    else:
        myOut += "--\t--"
    if ( i < len(x_out) ):
        myOut += "\t" + str(x_out[i]) + "\t" + str(y_out[i]) + "\t" + str(y_fit[i]) + "\t" + str(y_env_eval[i]) + "\t" +
str(y_s_out[i])
    myOut += "\n"
    i+=1
myFile = open(OUTPUT_DIRECTORY+"ms_emit.dat","w")
myFile.write(myOut)
myFile.close()

print sdt() + " -- Generating plots"
call("./usr/bin/gnuplot " + OUTPUT_DIRECTORY + "ms_emit.gnuPlot", shell=True)

print sdt() + " -- Syncing data to www-bd"
call("./usr/bin/wget -O /export/home1/edstrom/acl/emit_calc/ms_emit.sync -o /export/home1/edstrom/acl/emit_calc/ms_emit.wget
http://www-bd.fnal.gov/cgi-srf/fast.sync.pl", shell=True)
else:
    print sdt() + " -- Not Syncing to www-bd"

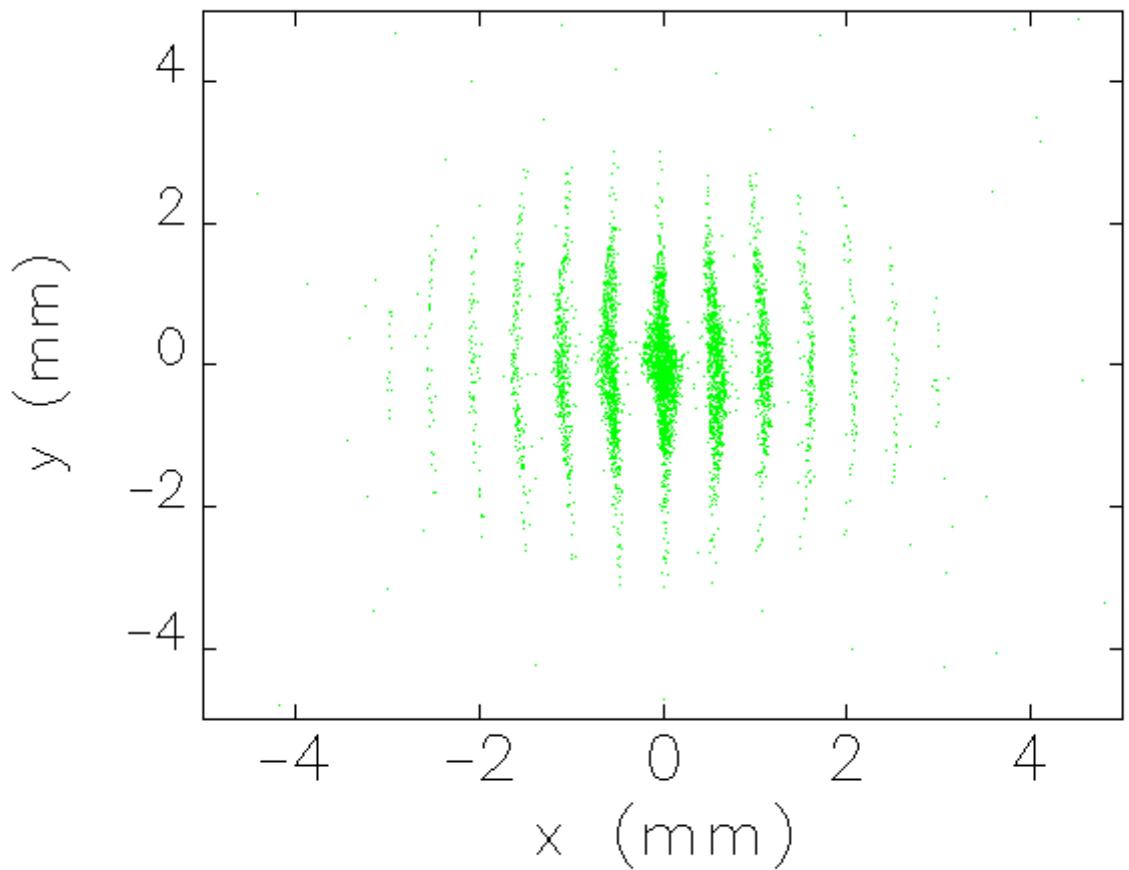
if(READ_LIVE):
    print sdt() + " -- Output to ACNET"
    tnow = (now() - t0)
    result = set_output([myEmitum, myAlpha, myBeta, myGamma, n_bumps, noise_level, pick_threshold, full_chisq, tnow])
    tar = tarfile.open(ARCHIVE_DIRECTORY + "ms_emit." + filedate + ".tar", "w")
    tar.add(OUTPUT_DIRECTORY + "ms_emit.htm","ms_emit.htm")
    tar.add(OUTPUT_DIRECTORY + "ms_emit.png","ms_emit.png")
    tar.add(OUTPUT_DIRECTORY + "ms_phase.png","ms_phase.png")
    tar.add(ARCHIVE_DIRECTORY + "fast_ms_emit.hist","ms_emit.hist")
    tar.close()

print sdt() + " -- DONE!"

if (DEBUG):
    y_fit = pevalb_full(x, flatten_array(a))
    print sdt() + " -- Plotting Summary..."
    plot_arrays([y_env_eval,y_s,y,y_fit],[x])
    print sdt() + " -- Plotting phase ellipse..."
    plot_arrays([y_ellipse],[x_ellipse])
quit()

```

Appendix C – Full Resolution Image of 3200_min_dE.X110.png



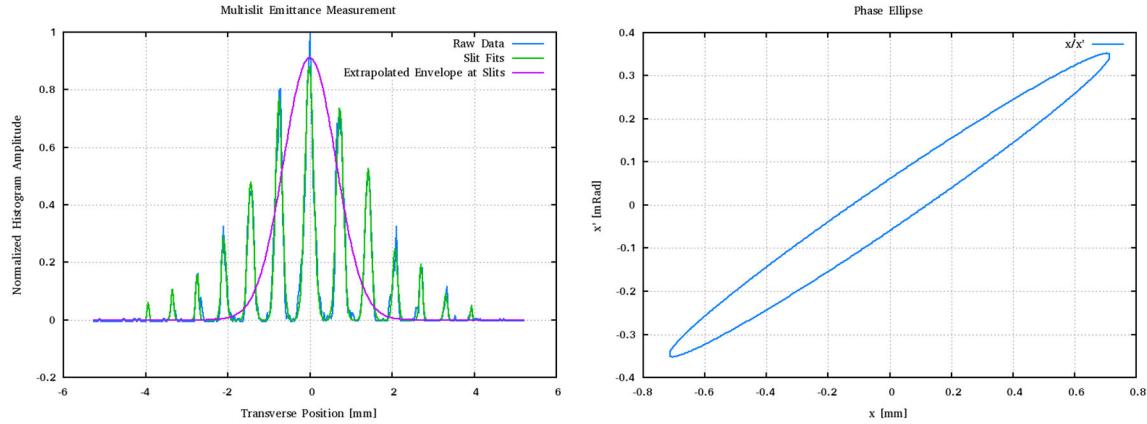
Appendix D – Example fast_ms_emit.py output to http://www-ad.fnal.gov/fast/sync/ms_emit.htm

Multislit Emittance Summary

File: sample.dat (10:36:29 02-14-2018)

16:54:38 02-14-2018

Emittance & System Params:		Second Moments:		Twiss Parameters:	
Emittance (Un-Norm) :	4.304e-02 μm	$\langle x^2 \rangle$:	5.126e-07 m^2	α :	-5.772
Dist to Screen (L) :	1.538e+00 m	$\langle x^{\prime 2} \rangle$:	1.240e-07 rad^2	β :	11.910
Slit Width (w) :	4.000e-05 m	$\langle xx' \rangle$:	2.484e-07 m^2rad	Y :	2.881
Slit Spacing (s) :	4.000e-04 m	Checks:		Background Subtraction:	
Histogram Amplitude :	1.00 ArbU	Full x2 :	8.033e-04	Gaussian Fit	
Histogram Width :	366 px	σ_{Env2} :	4.072e-07 m^2	Full Final Fit:	
Camera Resolution :	2.876e-05 m/px	$\langle x^2 \rangle / \sigma_{\text{Env2}}$:	1.259	--> 56 Parameters	



13 Gaussians Detected (Threshold = 0.115 | Noise = 0.005)

Gaussian	Amplitude	Centroid [m]	Sigma [m]	LinOff	Integral [m]	Div [Rad]
1	6.275e-02	-3.936e-03	2.229e-05	--	9.265e-17	-9.945e-04
2	1.179e-01	-3.348e-03	2.979e-05	--	8.806e-06	-8.721e-04
3	1.674e-01	-2.741e-03	3.655e-05	--	1.533e-05	-7.377e-04
4	2.904e-01	-2.097e-03	5.713e-05	--	4.159e-05	-5.791e-04
5	4.800e-01	-1.439e-03	7.386e-05	--	8.887e-05	-4.115e-04
6	7.747e-01	-7.532e-04	8.203e-05	--	1.593e-04	-2.255e-04
7	8.872e-01	-1.182e-05	1.031e-04	--	2.293e-04	-3.538e-06
8	7.410e-01	7.297e-04	8.621e-05	--	1.601e-04	2.185e-04
9	5.296e-01	1.418e-03	7.488e-05	--	9.940e-05	4.058e-04
10	2.501e-01	2.079e-03	5.975e-05	--	3.746e-05	5.756e-04
11	1.941e-01	2.705e-03	3.860e-05	--	1.878e-05	7.229e-04
12	9.312e-02	3.306e-03	3.984e-05	--	9.299e-06	8.535e-04
13	5.408e-02	3.930e-03	2.288e-05	--	9.163e-16	9.992e-04
Baseline	1.674e+01	1.573e+02	5.950e+02	-1.616e+01	-2.050e-05	
Proj Env	9.100e-01	0.000e+00	6.381e-04	1.045e-01	1.456e-03	